Naval Research Laboratory

Washington, DC 20375-5000

NRL Memorandum Report 6868

CLESIM: A Simulation Model of the Conventional Link-11 Roll-Call Network

DONALD G. KALLGREN AND LAM N. PAM

Communication Systems Branch
Information Technology Division

August 10, 1991

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	1991 August 10	Final

4. TITLE AND SUBTITLE
CLESIM: A Simulation Model of the Conventional Link-11 Roll-Call Network

5. FUNDING NUMBERS
55-2372-0-1

6. AUTHOR(S)
Donald G. Kallgren and Lam N. Pam

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Research Laboratory
Washington, DC 20375-5000

8. PERFORMING ORGANIZATION REPORT NUMBER
NRL Memorandum Report 6868

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Space and Naval Warefare Systems Command
PMW-159 Tactical Data Links
Washington, DC

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This memorandum describes a simulation model used to evaluate the current Link-11 (CLE) tactical data network. The model was developed to compare the performance of the network when operated with different modems, e.g., parallel-tone or single-tone modems. The simulation uses a finite-state model of the roll-call protocol and models transient behavior in the network resulting from channel errors. The model calculates net cycle time, channel utilization, injected traffic rate, and normalized throughput.

14. SUBJECT TERMS			15. NUMBER OF PAGES
Link-11, Roll-call, Tactical data networks		TADIL A Simulation	31 16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAR

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18
298-102

CONTENTS

FIGURES

Tables

CLESIM: A SIMULATION MODEL OF THE CONVENTIONAL LINK-11 ROLL-CALL NETWORK

1. INTRODUCTION

This memorandum describes a simulation model used to evaluate the current Link-11 (CLE) tactical data network. The baseline description of the Link-11 networking protocols used in this model was obtained from reference (a), and is summarized in section 2 of this memorandum. The reader should already be familiar with the Link-11 system, its networking protocols, and its waveform. The original purpose of the model was to provide the capability to compare the performance of the Link-11 network when the network is operating with different modems. The model can accommodate both the currently used parallel tone modems and the new single tone modems proposed for Interim Improved Link-11 (reference (b)). The performance measures calculated in this model are the net cycle time, the percent channel utilization, the injected traffic rate and the normalized effective throughput.

A closed-form probabilistic model of the Link 11 Roll-Call protocol has been described in reference (c) . This model is appropriate for analysis of the Link 11 protocol when the same channel connects all network members, i.e., when the waveform performance is identical for all links in the network. The probabilistic model can be used to determine the average CLE network performance. Analysis of transient behavior, analysis for networks with different link performance, and analysis of second-order effects that occur when the protocol breaks down can be performed using the simulation model described here.

2. CURRENT LINK-11 NETWORKING PROTOCOL (ROLL CALL)

A representative Link-11 Net is shown in Figure 1.The channel access protocol used in CLE is based on a centralized network control architecture. One node of the network, designated as the Data Net Control Station (DNCS), controls the channel access of all other nodes in the network. All the other nodes are called picket stations, or Participating Units (PUs), and can only transmit information into the channel when prompted to do so by the DNCS. A summary of the automatic interrogation of the pickets as described in reference (a) is summarized here.

The DNCS polls each picket station of the network in the order established by an address generator or by the TDS computer. The interrogation of the DNCS is composed of a preamble, a phase reference, and a picket address. The picket station whose address was polled then transmits the following picket reply message:

a) Preamble and phase reference (i.e., the synchronization preamble)
b) Start-of-Message code
c) Any number of TDS message frames
d) Picket stop (i.e., end-of-message) code

If a DNCS does not recognize a valid reply from a picket (i.e. if it does not recognize a start code) within 15 frames after interrogation, the DNCS sends another interrogation to the same picket.

Manuscript approved June 14, 1991.

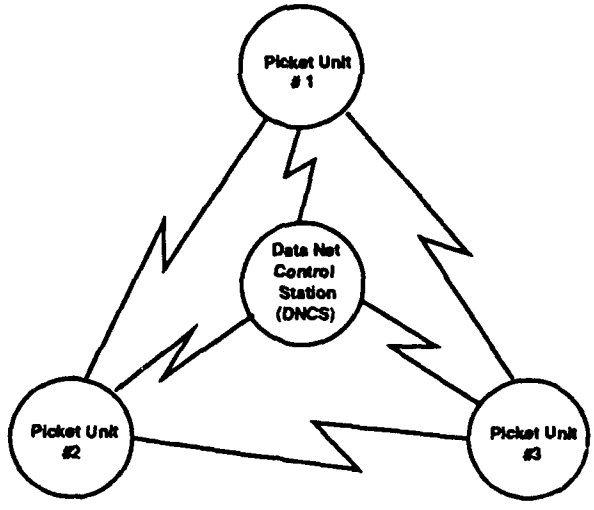

Figure 1. Representative Link-11 Network
(after MIL-STD-188-203-1A)

If the DNCS does not recognize a valid reply to the second interrogation within 15 frames, it shall interrogate the next picket.

If the DNCS receives a start code after either the first or second interrogation of a picket station, the DNCS will not interrogate the next picket until either of the following occurs:

a) A picket stop code is recognized
b) Loss of signal presence is determined

When the DNCS determines that it is its turn to transmit TDS data, the DNCS transmits a frame structure consisting of a preamble, a phase reference, a start code, any number of TDS message frames, a stop code, and the address code of the next picket to be interrogated.

The current Link-11 waveform is pictured in Figure 2. It consists of five parts. These parts are divided into frames, which are 13.3 ms long and consist of 30 bits of data (24 information bits and 6 parity bits). The preamble is 5 frames and is composed of two tones, an unmodulated 605 Hz tone for a Doppler tone and a 2915 Hz bi-phase modulated synchronization tone. The phase reference frame is composed of a 16 tone composite signal. The start of message code consists of 2 frames of known data. For the purposes of this study, an average data message was considered to be 48 information bits, or two frames. The end-of-message code is 2 frames of known data.

The maximum information transmission rate of the network is 1800 bits-per-second (bps); as a practical matter, the maximum throughput for nets with perfect waveform operation is more on the order of 1000-1300 bps, because of the interactions between the roll-call protocol performance, the number of nodes in the net, and the number of tracks reported by each node.

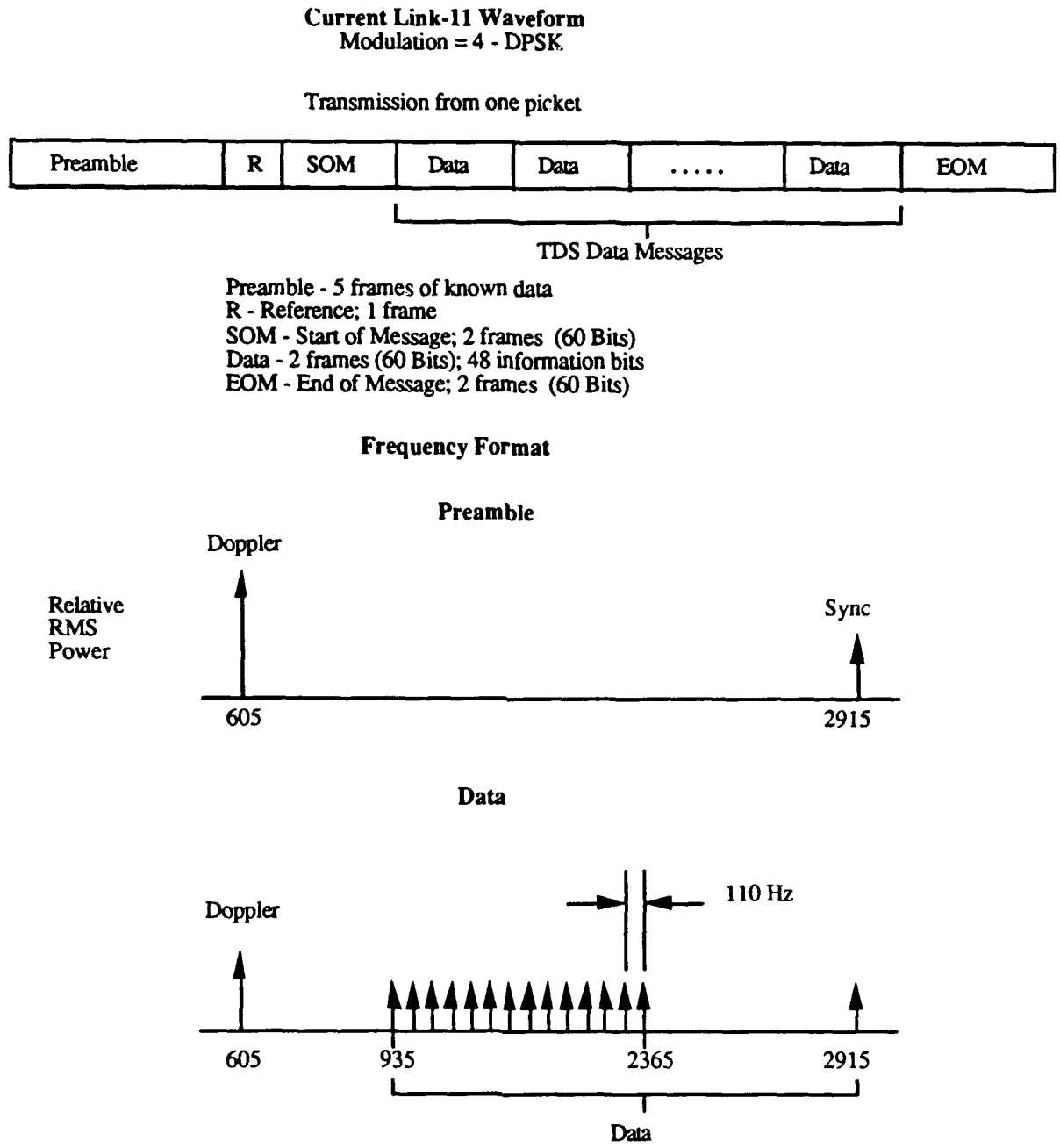

Figure 2. Link-11 Waveform Specification (Audio Baseband)

3 SIMULATION PROGRAM OVERVIEW

The program design language for the simulation is C. This version of the program has been run and tested on an Apple Macintosh II using Lightspeed™ C. While not tested in other environments, every attempt has been made to use routines and #include files commonly found in the standard UNIX programming environment; at present, all program I/O is supported using the UNIX command-line interface, stdin, stdout, and stderr.

CLE performance is computed using an event-driven simulation of a Finite-State Machine (FSM), representing the sequence of states that occur in the Link-11 Data Net Control Station (DNCS). For each node k, the DNCS can be performing one of the following processes:

- sending and waiting for a response to the first interrogation of node k
- sending and waiting for a response to the second interrogation of node k
- processing the reply from node k

A model of the DNCS FSM is shown in Figure 3. Transition probabilities in the simulation model are determined by the link and waveform characteristics defined for the scenario being simulated Using the transition probabilities and a random number generator, the simulation determines which of the three paths will be taken to complete the roll-call interrogation of a given PU. Statistics are computed as a function of the state transitions and reported at the conclusion of the simulation run.

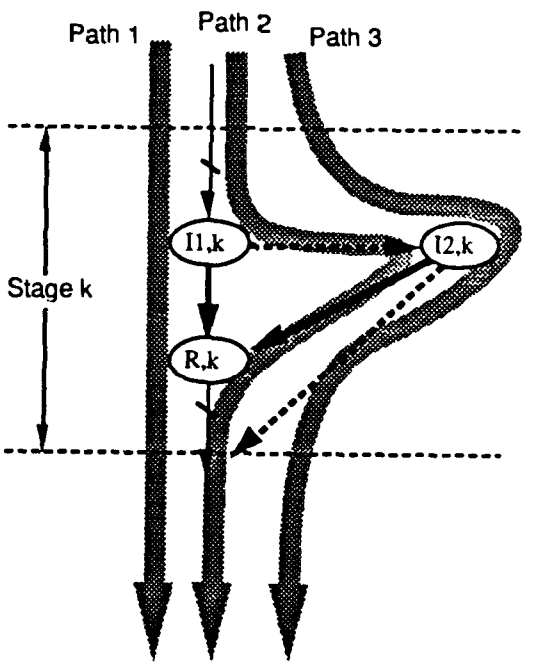

Figure 3. Finite State Machine (FSM) Representation of the Current Link-11 Roll-Call protocol.

3.1. Representation of States

States are stored and accessed in a typed data structure, represented (using C as the design language) as follows:

```
typedef struct
    {
    long int ncf;
    int type , pu, collision,tds_msgs_sent,
                                missed_eom,missed_reply;
    double  collis_end;
      } state;
```

The <state.type> can be one of the three defined constants: *intero_1*, *intero_2*, and *reply*, representing, respectively, the first interrogation state, the second interrogation state, and the reply processing state. The state.pu can be any integer from 1 to *number_pu_s*, where *number_pu_s* <= the defined constant max_num_pu. The states will at times be denoted (I1,k), (I2,k), and (R,k), respectively. So far, this is identical to the probabilistic model in reference (c), and we have simulated the roll-call protocol performance without collision effects as a cross-check on the accuracy and performance of both models, confirming each model. The additional fields in the data structure are used to denote special events that may have occurred during the state, such as a missed-reply, a collision between an interrogation and reply, the number of messages sent by the tactical data system, and whether the exit from the reply state occurred because of an EOM detection or a missed EOM.

The FSM maintains two state variables, current_state and next_state; it uses a procedure *get_next_state()* to compute the next_state from the current_state. This procedure uses the input arrays that define the probabilities of preamble synchronization, start-of-message detection, and end-of-message detection; these probabilities define the state-transitions in the absence of missed-reply or collision effects, and are used to cross-check the performance of the probabilistic and simulation models. When collision-detection is turned on in the simulation, however, *get_next_state()* uses the auxiliary state variables (i.e., *missed_reply*, *collision*, et. al.) as well as the primary state variable to determine the next state.

3.2. Computation of State Transition Probabilities

The state transition probabilities are computed using waveform performance data for each link that is provided as an input specification of the scenario. These probabilities are computed by the procedure *compute_transition_probabilites()* using the waveform performance parameters stored in the two-dimensional arrays *prob_sync[i][j]*, *prob_som_detect[i][j]*, *prob_eom_detect[i][j]*, and *prob_add_detect[i][j]*. These arrays correspond, respectively, to the waveform probability of synchronization, probability of detection for the start-of-message code and end-of-message code, and the probability of address detection, for the link from node i to node j.

State transition probabilities are stored in the array *state_trans_prob[k][l][j]*; the size of this array is 3 x 3 x *max_num_pu* , and the k,l,j entry is the probability of transition from state.*type* k to state.*type* l for node j. Transition probabilities are computed according to the

following rules, which assume independence between stages, no memory (this is a markov process !), and no collisions between transmissions [1] :

- ***transition probability from [intero_1,pu_ID] to [reply,pu_ID]***

  trans_prob[intero_1][reply][pu_ID] = prob_sync[dncs_ID][pu_ID] *
  prob_add_detect[dncs_ID][pu_ID] *
  prob_sync[pu_ID][dncs_ID] *
  prob_som_detect[pu_ID][dncs_ID];

- ***transition probability from [intero_2,pu_ID] to [reply,pu_ID]***

  trans_prob[intero_2][reply][pu_ID] = trans_prob[intero_1][reply][pu_ID];

- ***transition probability from [intero_1,pu_ID] to [intero_2,pu_ID]***

  trans_prob[intero_1][intero_2][pu_ID] = 1 - trans_prob[intero_1][reply][pu_ID];

- ***transition probability from [intero_2,pu_ID] to [intero_1,(pu_ID+1)mod N]***

  trans_prob[intero_2][intero_1][pu_ID] = 1 - prob_sync[dncs_ID][pu_ID] *
  prob_add_detect[dncs_ID][pu_ID] *
  prob_sync[pu_ID][dncs_ID] *
  prob_som_detect[pu_ID][dncs_ID] ;
  = 1- trans_prob[intero_1][reply][pu_ID];

- ***transition probability from [reply,pu_ID] to [intero_1,(pu_ID+1)mod N]***

  trans_prob[reply][intero_1][pu_ID] = prob_eom_detect[pu_ID][dncs_ID] +
  some other stuff that depends on the performance
  of the signal-loss detection circuit;
  = 1; (i.e., we assume a perfect exit from the reply
  processing state to the interrogation of the next !);

NOTE: ALL OTHER TRANSITION PROBABILITIES IN THE MODEL ARE ZERO.

3.3. Analysis and Statistics Collection

The general program flow in the CLE simulation is as follows:

```
initialize();
get_program_control_input();
get_analysis_input_parameters();
do_analysis();
do_results();
```

[1] When collisions between transmissions are modelled, transient modifications to the state are generated and stored in the auxiliary fields of the state variables ; no permanent changes in the state-transition probabilites are computed for collision events.

Do_analysis() is the key top-level procedure. Basically, *do_analysis()* is a counting process aimed at determining the total number of preamble frames, phase-reference frames, start and stop codes, and message frames sent during the net cycle; the counting loop also tracks the total time that the channel is idle for any reason, whether for propagation delays or time waiting for a reply to the interrogation. From these calculations, the procedure computes the total time spent in traversing the state space. The net cycle time is defined as the time it takes between successive entries into state (I1,1). The procedure keeps a running estimate of the mean and variance of the network cycle time, as well as estimates of the channel utilization for each contributor to channel usage, and the injected traffic rate. The counter processing is modified when collision detection is turned on, to track the traffic injected by missed replies, and to monitor channel occupancy. The analysis sequence is as follows:

```
compute_transition_probabilities();
current_state = init_state();
next_state = init_state();
while (not done)
    {
        next_state = get_next_state();      /* some of the calculations ***/
                                            /* may depend on the next state */
        count_transmissions();     /** do all the bookkeeping *****/
        count_syncs_sent();
        count_addresses_sent();
        count_som_sent();
        count_eom_sent();
        count_mi_sent();
        count_msgs_collided();
        count_tds_msgs_sent();
        count_tds_msgs_rcvd();
        count_interrogations();
        count_successful_interrogations();
        count_collisions();
        count_idle_time();
        advance_clock();
        update_scen();
    }
```

Results are produced as message-trace listings, and tabular text summaries of statistical performance. Of the various forms of output that can be selected using command-line arguments, only the message trace and tabular outputs are fully functional; none of the alternate graphic output forms for which command-line selectors have been defined are functional at present.

A synopsis of the key procedures in the simulation follows.

3.3.1. Determination of the Next State in the Roll-Call Protocol

The procedure *get_next_state()* determines the next state in the roll-call protocol, based on the current state and the state-transition probabilities. The routine draws a number from a uniform distribution over the interval [0,1], compares it to thresholds defined by the array *state_trans_prob[][][]*, and uses the comparison to determine the next state. If the transition is from *reply* to *intero_1*,then the function sets *the_next_state.pu = (current_state.pu + 1)* modulo the number of nodes in the net; it also skips the states where the DNCS would be interrogating itself. Note that the interrogation sequence is fixed in this model; nodes are interrogated in numerical order by PU number, and the DNCS is always the first node, node 0, in the list.

If collision detection is turned off, the state variables are set by one of the procedures called by *get_next_state()* : *normal_intero1()*, *normal_intero2()*, *normal_reply_state()*, or *intero1_missed_eom()*.

If collision detection is turned on in the model, then the state variables may be set additionally by the procedures *int1_w_col()* or *int2_w_col()*, depending on the current state and any of the auxiliary state variables <state.*missed_reply*, state.*collis_end*>. These additional procedures generate transient modifications of the auxiliary variables state.*collision*, state.*missed_eom*, or state.*missed_reply*. These variables will modify the state sequence and the statistics collection until the collision event terminates. After the collision event, state transitions are defined as above. A detailed analysis of collision events is presented below.

3.3.2. Analysis of Collision Effects in the Roll-Call Protocol.

The probabilistic model of the Link-11 Roll-Call protocol described in reference c) has several advantages. It is a closed-form model that provides results readily for a variety of network performance parameters, Also, it incorporates several key waveform performance parameters that permit study of modem tradeoffs and their effects on network performance. The probabilistic model, which is the basis for the default operating mode in this simulation, has some assumptions and limitations that merit examination and discussion. All of these assumptions and limitations are based inherently on the underlying model of the roll-call protocol as a process in which the probability of state transitions depends solely on the current state, i.e., we have assumed that the roll-call protocol is a Markov process.

The first of the assumptions is that stages in our model are independent. The path (and performance metric) for one stage does not affect the path/performance of succeeding stages. The roll-call protocol has no memory of the path taken to reach a given state and, other than knowledge of whether it is performing the first or second interrogation of a picket unit, has no memory of past success or failure in its interrogations. This means that the protocol does not, for instance, modify the polling cycle based on the interrogation success rate of each picket unit. Under the current Link 11 specification, such modification could be performed by the tactical data system (TDS) using some unspecified method, with the Link 11 data terminal set accepting its picket address for each interrogation from the TDS. Our model is invalid for such a possible mode of operation.

The second assumption is that there are no false alarms in the system, e.g., a picket unit will not declare an address detection erroneously and proceed to transmit a reply. A false-synchronization and reply is considered to be an extremely unlikely event, and is excluded from our model. Of more concern is the possibility of false alarm for end-of-message detection, or, actually, premature declaration of end-of-message. A premature end-of-message

declaration would result in lost TDS messages in the reply; if the DNCS were the unit that prematurely declared end-of-message, it would send an interrogation to the next picket unit in the poll with the possibility of a collision with the reply. We have assumed that the probability of false declaration of end-of-message is zero.

The most critical assumption that has been made is that there are no simultaneous transmissions on the channel that would result in a collision. However, collisions can occur in the Link-11 roll-call protocol when a PU responds to an interrogation, but the DNCS fails to detect the reply start code (including the preamble and phase reference). In this case, the DNCS will wait for the missed-reply timeout period, and transmit another interrogation.

Collisions will change the transition probabilities in an actual roll-call system. For example, if a missed-reply is of even moderate length (typically, greater than two M-series messages) it will still be in transmission when the DNCS protocol controller sends its interrogation. The probability of this interrogation failing is less dependent on waveform parameters than it is on the fact that it is colliding with a reply. In a high signal-to-noise channel, the probability of interrogation failure after a collision is effectively one, *conditioned on the failure of the preceding interrogation.* If the reply is of moderate length (typically, 15 M-series messages) then the first interrogation that collides will be followed by a second interrogation that will also collide with the reply, etc. Clearly, the independence and no-memory assumptions are valid only if the probability of collisions is very small, ideally zero.

Unfortunately, the probability of a collision is not zero, or even very small. It is a function of the same waveform performance parameters that have been included in the probabilistic model. Calculation of the collision probability is straightforward, and equals :

$$P_{collision}[k] = P_{sync}[d][k]*P_{add}[d][k] - P_{sync}[d][k]*P_{add}[d][k]*P_{sync}[k][d]*P_{som}[k][d]$$

The relationships between the probabilities of missed-interrogation, successful interrogation, and collision are illustrated in Figure 4.

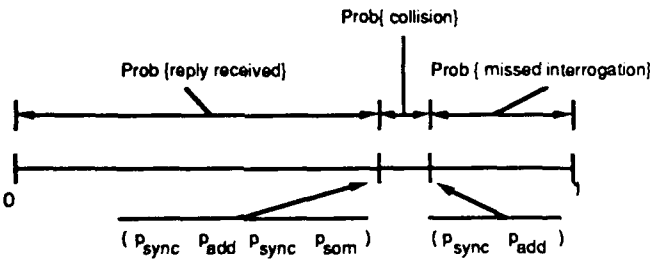

Figure 4 - Relationships between waveform performance parameters and the probabilities of missed-interrogation, successful interrogation, and collision

The dependence of the probability of collision on the synchronization and address-detection performance of the waveform is illustrated in Figures 5 and 6. For perfect address-detection and SOM-detection, and a symetric channel between the DNCS and the PU, the probability of collision is at most 0.25, occurring when the probability of synchronization is one-half; as the probability of SOM-detection degrades, the probability of collision can increase, however.

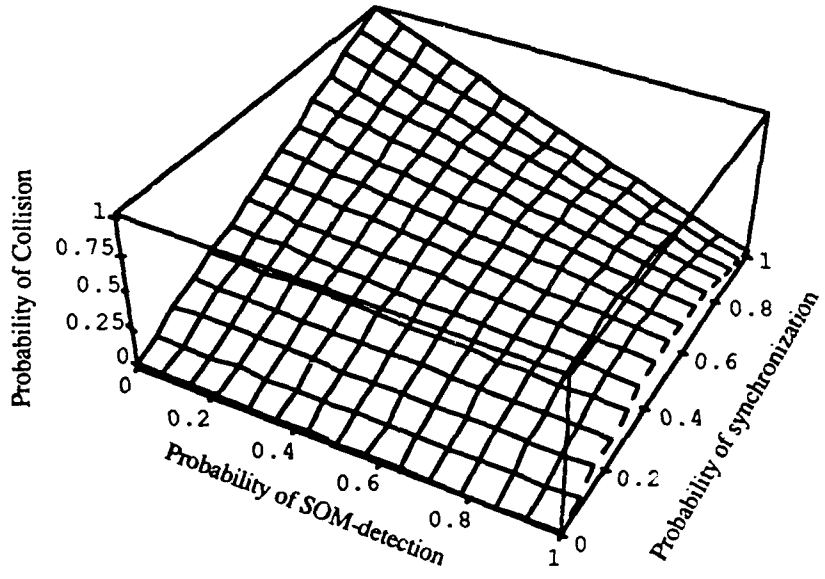

Figure 5 . Probability of Collision as a function of synchronization and SOM-detection performance, (probability of address-detection is 1).

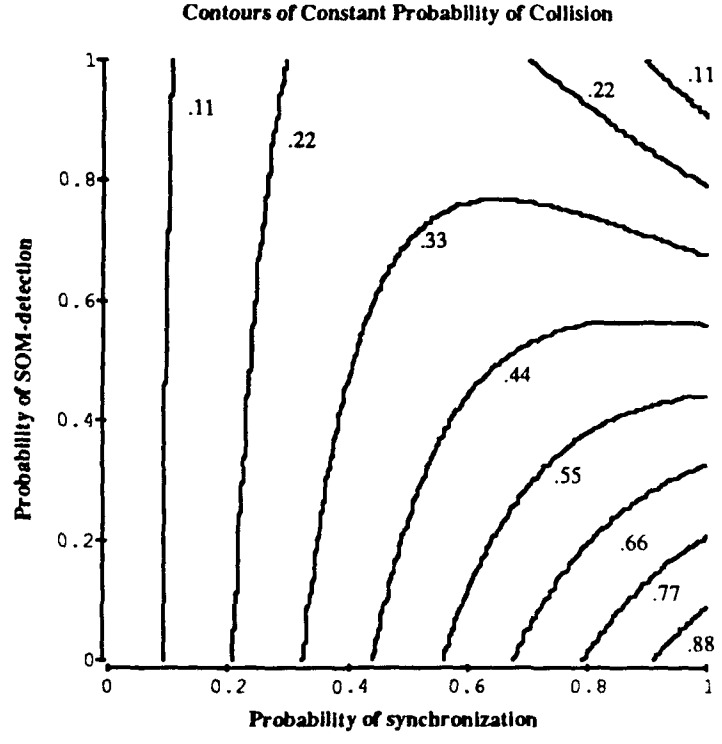

Figure 6. Contours of constant probability of collision, as a function of synchronization and SOM-detection performance, (probability of address-detection is 1).

In the simulation, whenever collision detection is selected as a command-line option, these relationships are used by the procedure *get_next_state()* to determine if the reply was missed and, on that basis, to modify the polling sequence.

The polling sequence is modified for the duration of the collision. The procedure *end_of_collision_time()* accepts as its input the number of tds messages sent in the reply involved in the collision, and computes the time at which the collision ends, i.e., it computes the time at which the reply will cease transmission. Based on the reply end-of-transmission, the type of collision is determined. The following analysis is the general form for determining the collision type and its effects; at present, collisions are computed only at the DNCS and the PU being interrogated. Since the propagation delays are generally much less than the durations of replies, interrogations, and timeouts, this is considered a reasonable approximation to the collisions that would occur at any PU. The simulation will require some modification to calculate collisions exactly and independently at each PU, however. For either case, the analysis of the collision type is based on the following.

The simulation distinguishes between three types of collisions: fatal, interior, and tail-end. A fatal collision is one in which all data in the reply is lost, and occurs because the erroneous interrogation corrupts the modem and/or crypto synchronization preamble so that demodulating and decrypting the tactical data system (TDS) messages is impossible. An interior collision occurs when the erroneous interrogation is received after a node has synchronized to the reply and started to decrypt TDS messages, and the interrogation ceases before the reply end-of-message (EOM) code. A tail-end collision occurs when the erroneous interrogation collides with some TDS messages and the EOM.

To simplify the analysis somewhat, a fatal collision occurs if the erroneous interrogation overlaps any portion of the reply preamble, phase reference, start of message, or the crypto preamble that is the first portion of the TDS message. In general, there may be some receiver capture mechanism or sufficient difference in signal strengths that might allow another PU to receive the reply, but complete loss of data cannot occur because of a collision event if these conditions are not met. We do not consider situations in which a fatal collision occurs and a PU is still able to correctly decode and decrypt the reply message. The timeline for a fatal collision is shown in Figure 7. For this figure and those which follow, we use the following notation:

d = the data net control station ID;
k = node k in the network;
j = node j in the network;
$T_{prop}(d,j)$ = the propagation delay from the DNCS to node j;
$T_{response}$ = PU response time from address recognition to transmission of reply ;
T_{sync} = duration of preamble and phase reference symbols;
T_{add} = duration of address code;
T_{som} = duration of the start-of-message code;
$T_{timeout}$ = missed-reply timeout duration at DNCS;
T_{MI} = duration of the crypto synchronization preamble;
T_{tds} = duration of a tactical data message

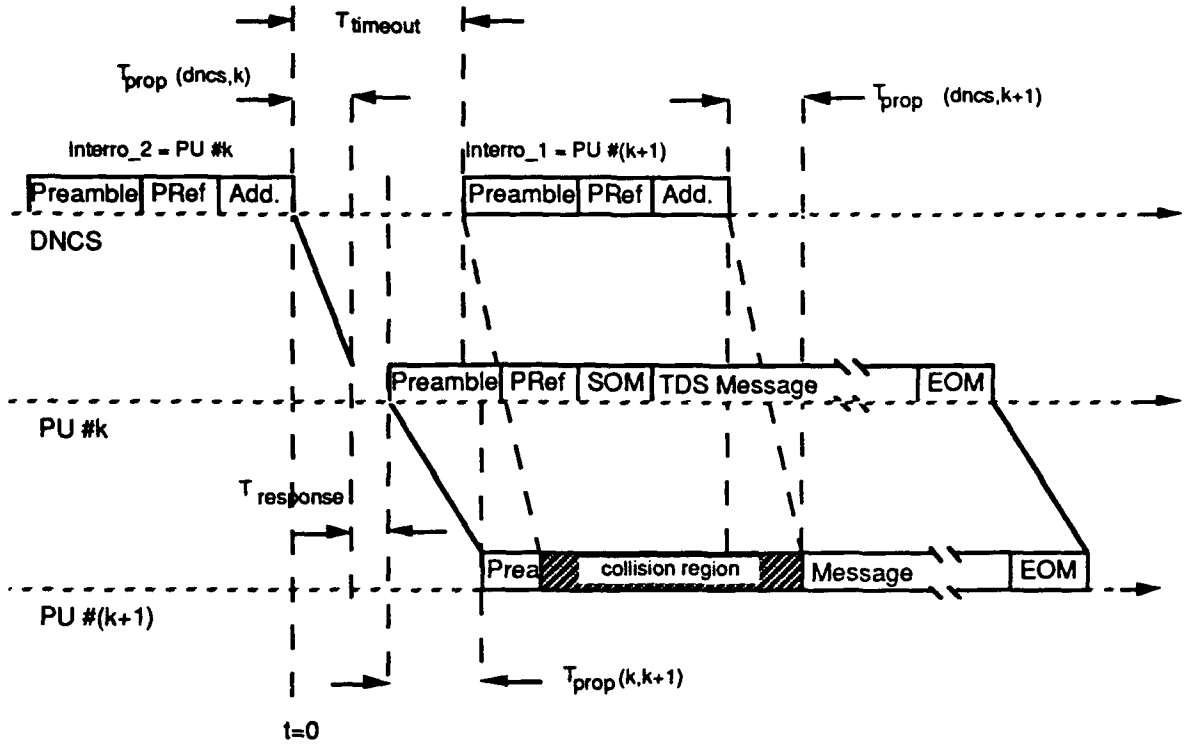

Figure 7 - Timelines for Link-11 collision event: fatal collision between missed PU reply and DNCS interrogation

Two boundary conditions must be satisfied for a fatal collision to occur at a given node. The first is that the interrogation preamble must arrive before the end of the crypto-synchronization preamble. The second is that the interrogation address code must arrive after the start of the reply preamble. Our assumption is that when these two conditions are satisfied, correct demodulation of the crypto synchronization preamble is impossible; without the crypto-synchronization preamble, decryption of the TDS messages in the reply is impossible and therefore all the TDS data is received in error.

An interior collision occurs when the reply crypto-synchronization preamble is correctly received, but the erroneous interrogation introduces errors into the TDS messages that comprise the body of the reply message. Our assumption is that only the TDS messages overlapped by the erroneous interrogation are received in error, and that this number will always be the same for all interior collisions since the interrogation length is always the same. An interior collision is illustrated in Figure 8.

The two boundary conditions that must be satisfied for an interior collision are that the interrogation preamble must be arrive after the end of the reply crypto-synchronization preamble, and that the interrogation address code must arrive before the reply EOM code.

The number of messages K lost because of the collision is the number of messages overlapped by an interrogation:

$$K = \left\lceil \frac{(T_{sync} + T_{add})}{T_{tds}} \right\rceil, \qquad \text{where } \lceil x \rceil \text{ is the smallest integer greater than x.}$$

A tail-end collision occurs when the interrogation preamble overlaps the reply EOM code. Our assumption here is that the number of TDS messages received in error is a function of the time-of-arrival of the interrogation preamble relative to the EOM. A tail-end collision is illustrated in Figure 9. The boundary conditions for this event are that the interrogation preamble arrives before the end of the reply and the interrogation address arrives after the EOM code.

The number of messages lost in a tail-end collision are the number of messages overlapped by the interval starting with the interrogation preamble and ending with the reply end-of-message. This number is

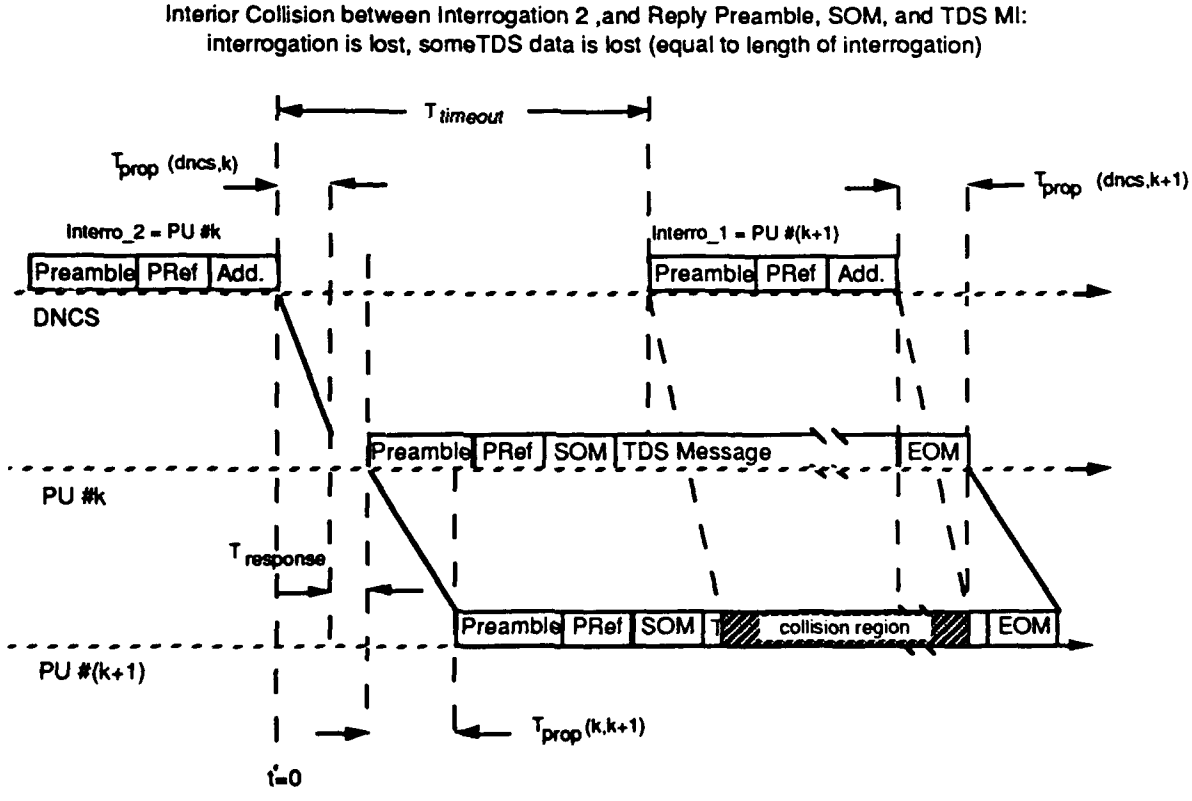

Figure 8 - Timelines for Link-11 collision event: interior collision between missed PU reply and DNCS interrogation

$$K = \left\lceil \frac{(T_{overlap})}{T_{tds}} \right\rceil, \quad \text{where } \lceil x \rceil \text{ is the smallest integer greater than x;}$$

$T_{overlap}$ is the portion of the reply starting at the arrival of the interrogation preamble and ending at the start of the EOM-code.

The collision analysis is summarized in Figure 10 and in Table 1. The type of collision is determined by comparing a parameter that depends solely on the roll-call protocol design and differential propagation delays for the collision scenario; the decision regions are defined solely by message parameters. Figure 10 illustrates the boundary conditions and collision region for each type. The number of messages lost in the collision depends on the type of collision, and is likewise dependent on the message parameters, protocol-design, and differential propagation delays.

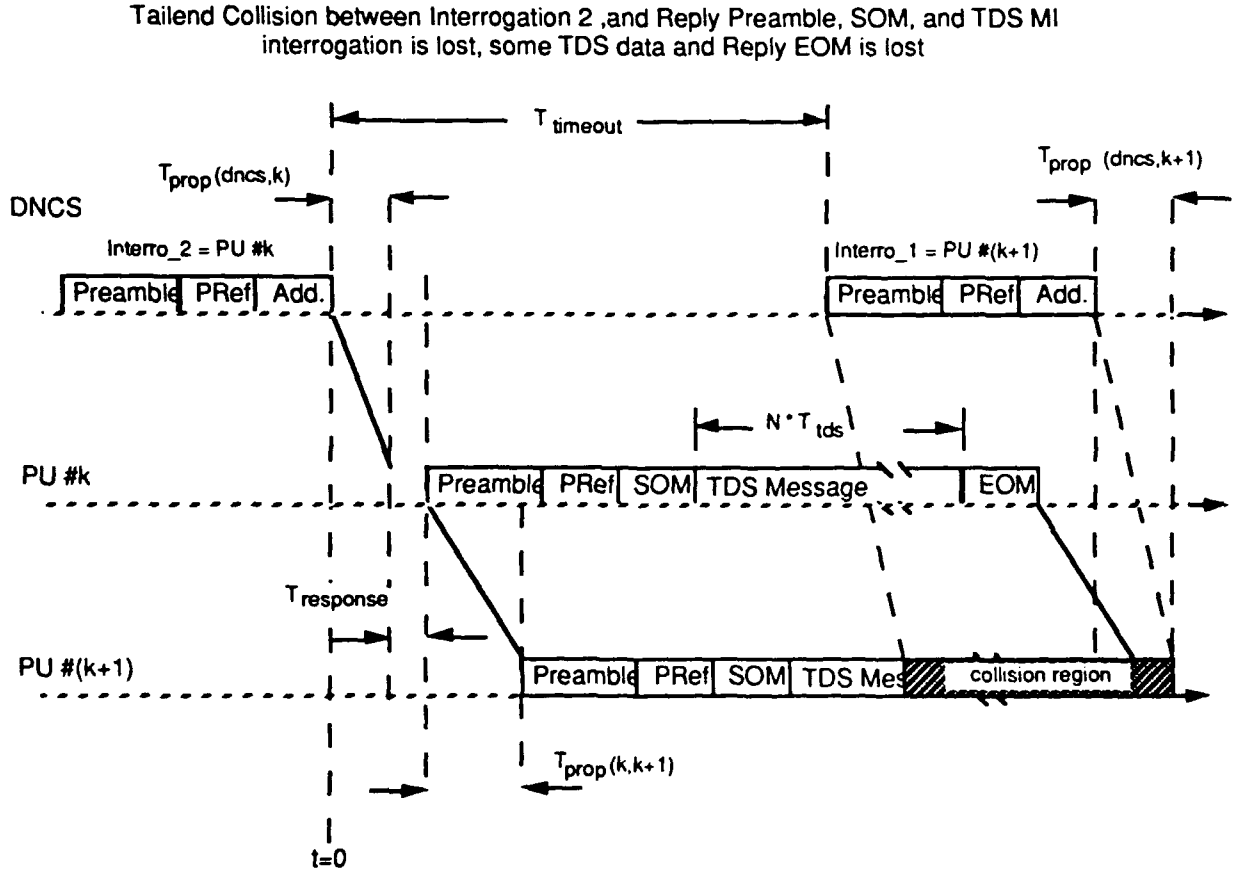

Figure 9 - Timelines for Link-11 collision event: tail-end collision between missed PU reply and DNCS interrogation

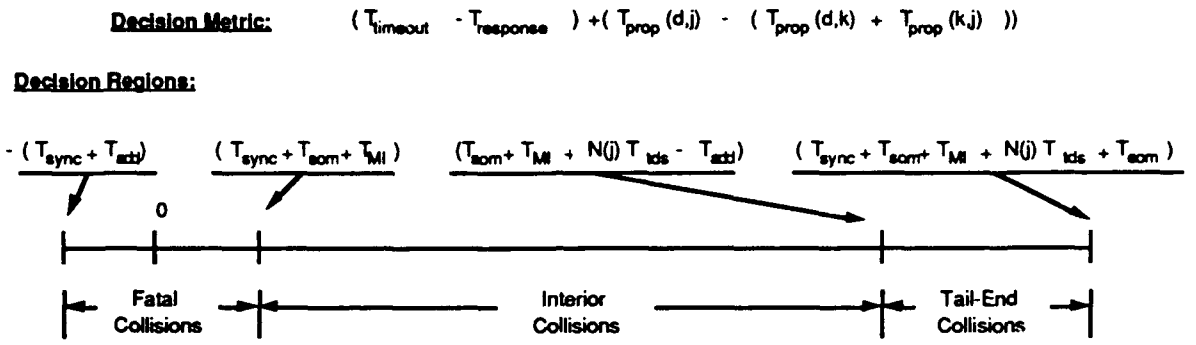

Figure 10 : Decision regions for collision-type determination

Collision Type	Number of Messages Lost
Fatal	N(j) ; where N(j) is the number of messages transmitted by node j
Interior	$K = \left\lceil \frac{(T_{sync} + T_{add})}{T_{tds}} \right\rceil$ 1)
Tail-End	$K = \left\lceil \frac{(T_{overlap})}{T_{tds}} \right\rceil$ 2)

Note 1: , where $\lceil x \rceil$ is the smallest integer greater than x;

Note 2: $T_{overlap} = (T_{sync} + T_{som} + T_{MI} + N(j)T_{tds}) - (T_{timeout} - T_{response}) + (T_{prop}(d,j) - (T_{prop}(d,k) + T_{prop}(k,j)))$

Table 1: Message-loss computation versus collision type

With the preceding as a theoretical basis, the procedure *count_msgs_collided()* uses these relationships to determine the number of messages lost to collisions during the roll-call protocol. The type of collision is computed by the procedure *collision_type()*, and stored in the auxiliary field <state.*collision*>, as one of the defined constants <*fatal, interior, tailend, none*>. The other counting routines in the program (e.g., *count_syncs_sent()*, *count_som_sent()*, etc) account for the additional message components that are transmitted during collision events.

3.3.3. *Raw Statistics Collection*

Raw statistics in the model are collected in several variables. One set is used for computing statistics for each net cycle, another set is used for computing cumulative statistics over the entire simulation run. The net-cycle raw statistics are the following:

```
int preambles_sent;
int phase_refs_sent;
int som_sent;
int eom_sent;
int addresses_sent;
int mi_sent;
int dncs_msgs_sent;
int tds_msgs_sent;
int msgs_collided;
long int tds_msgs_rcvd;
int replies_sent;
int num_interrogations;
int successful_interrogations;
int num_collisions;
int num_lost_msgs;
```

A slightly different set of raw statistics are accumulated over the entire simulation run:

```
long int number_of_transmissions;
long int tot_interrogations;
long int tot_successful_interrogations;
long int tot_collisions;
long int tot_lost_msgs;
double total_preamble_duration ;
double total_phase_ref_duration;
double total_start_stop_duration;
double total_address_duration ;
double total_dead_time ;
double total_data_duration ;
```

3.3.4. *Computation of Interrogation Success Rate, Traffic Injection Rate, and NCF Duration*

These routines compute the summary performance statistics for traffic injection rates, interrogation success rate, and net cycle duration:

```
compute_avg_success_rate()
compute_var_success_rate()
compute_avg_inject_rate()
compute_var_inject_rate()
compute_avg_ncf_duration()
compute_var_ncf_duration()
```

Statistical summaries are computed using cumulative values and mean-square values accumulated during the simulation run. These routines are straightforward computations using the raw statistics collected during the simulation run.

3.3.5. *Computation of Channel Utilization Statistics*

The following routines are used to collect channel-utilization statistics for each net cycle:

compute_ncf_dhama_eq_gb_pre_util()
compute_ncf_dhama_eq_dbhdr_util()
compute_ncf_dhama_eq_net_mngmnt_util()
compute_ncf_dhama_eq_inject_util()

These routines are straightforward, and compute channel utilization factors that may be compared to those generated by the Link-11 Improvement Simulation Model (LEISIM), written by the Rockwell-Marconi Joint-Venture Team. LEISIM generates channel utilization statistics for the Dynamic Handoff Assigned Multiple Access (DHAMA) protocol (reference (b)). For comparison purposes between the performance parameters for this model and for LEISIM, the following definitions are applied:

- *ncf_dhama_eq_gb_pre_util* is the channel utilization in the roll-call protocol incurred by the preamble, phase references, and the times when the channel is idle; the statistic is generated for each net cycle frame by taking the ratio of the sum of the duration of all preamble and phase references sent, plus the channel idle time, to the duration of the net cycle frame.
- *ncf_dhama_eq_dbhdr_util* is the channel utilization in the roll-call protocol incurred by the start code and the stop code; these codes are considered the closest analog in the Link-11 roll-call protocol to the data-block headers used in DHAMA. The statistic is generated for each net cycle frame by taking the ratio of the sum of the duration of all start and stop codes sent to the duration of the net cycle frame.
- *ncf_dhama_eq_net_mngmnt_util* is the channel utilization in the roll-call protocol incurred by the address codes sent by the DNCS when interrogating PUs; these codes are considered as the closest analog to the network management traffic generated by the DHAMA protocol. The statistic is generated each net cycle frame by taking the ratio of the sum of the duration of all addresses sent to the duration of the net cycle frame.
- *ncf_dhama_eq_inject_util* is the channel utilization in the roll-call protocol incurred by all tactical data messages sent (i.e., injected) into the network; TDS MESSAGES THAT ARE LOST BECAUSE OF A COLLISION BETWEEN A REPLY AND AN INTERROGATION ARE NOT INCLUDED IN THIS STATISTIC. The statistic is computed at the end of the net cycle frame by taking the ratio of the duration of all tds messages injected during a net cycle frame to the duration of the net cycle frame.

A similar set of routines is used to compute the cumulative channel utilization statistics:

compute_cumm_dhama_eq_gb_pre_util()
compute_cumm_dhama_eq_dbhdr_util()
compute_cumm_dhama_eq_net_mngmnt_util()
compute_cumm_dhama_eq_inject_util()

These routines are similar to the routines that compute the ncf utilization statistics. Similar definitions for each statistic are defined, but using counters that accumulate data for the entire simulation, rather than for one net cycle.

3.3.6. Event Tracing During Simulation

The procedure *do_trace()* is the central calling point for all tracing done within the simulation. The procedure is called with a message parameter that defines the type of trace that will be performed. If the message tracing is turned off, then this procedure exits immediately, otherwise, it tests the message passed to it and calls the appropriate trace procedure. The following trace procedures are defined:

- *do_s_matrix_rpt()* prints a report of the non-zero transition probabilities in the state matrix.
- *do_s_trace_rpt()* prints a report to stdout describing the current state.
- *do_transmission_rpt()* sends a report to stdout that describes the transmission, in a format that gives the preamble, phase ref., start code (if present), address (if present), tds messages (if present, it gives the number transmitted), and stop code (if present). Presence or absence of the various components of the transmission is determined by the state passed to the procedure when it is called.
- *do_collision_rpt()* sends a report to stdout on a collision event.
- *do_missed_eom_rpt()* sends a report to stdout on a missed-eom event.
- *do_ncf_summary_rpt()* prints a report describing the net cycle frame performance to stdout. The performance of the network cycle frame is maintained in the following variables: int preambles_sent; int phase_refs_sent; int som_sent; int eom_sent; int addresses_sent; int tds_msgs_sent; int replies_sent; int num_interrogations; int successful_interrogations; double channel_idle_time;. The values of these variables are added to the cumulative statistics at the end of each ncf and then reset to zero (by another routine). This routine may be called at any time.
- *do_transition_rpt()* prints a report to stdout describing the state transition, giving the old and new states.
- *do_missed_reply_rpt()* prints a report to stdout on a missed-reply event.

Event tracing during simulation is normally turned off. It can be turned on at program invocation by the command-line option -m.

3.4. Selecting Program Operating Modes

Various program operating modes are selectable by specifying command-line options at program invocation. The procedure *get_program_control_input()* parses the following command line arguments to set up the appropriate control switches for the simulation:

-m generates message tracing during program execution. The program uses the control switch *m_trace*, with values (TRUE/FALSE) The default mode is to suppress message tracing.

-tab generate tabular output, with tab-separated columns of data sent to stdout. This format is suitable for input to *Cricket Graph, Excel, Kaleidagraph* or some other graphing program, to plot the output of multiple simulation runs (e.g., for a parametric or sensitivity analysis). NOTE: ALL PROMPTS FOR DATA INPUT ARE SUPPRESSED WHEN THIS MODE IS SELECTED, SINCE THEY WILL INTERFERE WITH THE TABLE FORMAT. This output mode is intended for use with command-line redirection of *stdin*, with the interactive input parameters contained in the file substituted for *stdin*. Otherwise, you need an excellent memory for the order and format of scenario input parameters.

-in {filename} accept scenario input data, such as synchronization probabilites, from the designated filename. When this flag is selected, all scenario definition data will be taken from the file; otherwise, the program defaults to a mode that prompts for scenario data (in simplified form). The file name is stored in *scen_file_name*; the program uses the control switch *get_scen_from_file*, with values (TRUE/FALSE).

-c command line switch to enable collision detection in the model.

-rseed {integer} argument to input a new random seed controlling the FSM simulation. The program uses the default value for the random seed if this control is not set. It is an error if the input cannot be read as an integer. There is no control flag set for this command line control; this procedure sets the new random seed directly.

-tek generate tektronix-4014-compatible graphic output. The program uses the control switch *tek_graphics*, with values (TRUE/FALSE) THIS SWITCH DOES NOT CURRENTLY CONTROL ANYTHNG, BUT WILL BE CORRECTLY PARSED.

-p {filename} generate postscript-compatible graphic commands output to a text file. The program uses the control switch *ps_graphics*, with values (TRUE/FALSE). The file name is stored in a character array (string) called *ps_filename*; THIS SWITCH DOES NOT CURRENTLY CONTROL ANYTHNG, BUT WILL BE CORRECTLY PARSED.

Any of the graphics output or tabular format commands override the message trace command; it is a command-line error if multiple graph- or tabular- format commands are selected and the program will abort gracefully.

NOTE: - The hyphen is a part of the command-line switch .

Each command-line switch can be used alone or in combination with other switches. When a combination of command-line switches is used, the switches can be in any order. When typing in a combination of command-line switches, use spaces to separate the switches.

3.5. Running the Simulator

As noted previously, this version of the program has been run and tested on an Apple Macintosh II using Lightspeed™ C, Version 3.01. While not tested in other environments, every attempt has been made to use routines and #include files commonly found in the standard UNIX programming environment; at present, all program I/O is supported using the UNIX command-line interface, stdin, stdout, and stderr.

The CLE simulation program is designed to run in two input modes: the interactive mode, and the batch-file mode. In the interactive mode, this program prompts the users for key input parameters required to run the finite state machine. In the batch-file mode, users have to type in the name of a file which contains all required parameters to run the program; the file format for the batch-mode input file is given in Appendix A. The input mode is selected using the command-line switches.

One method of generating a series of analyses in the interactive mode is worth noting here, and is based on the use of file redirection in the UNIX operating environment. A set of input parameters can be stored in a file, in the same order in which the parameters are requested

interactively by the program. This file can be supplied as input to the simulator at invocation by redirection of stdin.

Software Distribution

The simulator is distributed on a 3.5" double-sided 800 Kbyte disk for the Apple Macintosh. Software included in the distribution includes the following documents and applications:

README - a text file containing information about the distribution

cle.anal.main.π - the LightSpeed™ C project file for the simulation

cle.anal.main.c - the simulation source code

cle_mil_std.h - the header file containing defined constants for the Link-11 Military Standard; this file may be changed to examine performance changes that result if waveform and timing parameters are changed

psync.sens.test - a sample text file used to study the sensitivity of Link-11 network performance to changes in the probability of synchronization; this file is an example of the file structure required to run the simulator in interactive mode and redirecting stdin.

testin2.1 - a sample text file for running the simulator in batch mode using a predefined scenario file.

Link11_sim - a stand alone Macintosh Application that executes the Link 11 simulation.

Running the Simulation with Lightspeed™ C on the Apple Macintosh

To execute the simulation program in the Lightspeed™ C environment, these steps should be followed:

Step 1: Load the simulation program.

To load the program, double click on the file name (normally, this is *cle.anal.main.c*. A window which shows the project name is displayed. Double click on *cle.anal.main.π*. A window which contains the project is displayed.

Step 2: Run the simulation program.

- Select **RUN** from the **PROJECT** menu of THINK C. A new window will be displayed and a command "Enter Unix Command-Line" will be shown.

- To run in the interactive mode, type in any command-line switches (e.g., -m, -c, -rseed, or combination of them) depending on which feature that users want to use. Hit **RETURN** and the program will start to run. A series of prompts for waveform parameters will be generated unless the -tab option has been selected

- To run in the batch-file mode, type in *-in /filename/*, and any other command-line switches (e.g., -m, -c, -rseed, or combination of them depending on which feature that users want to use.) Hit **RETURN** and the program will start to run.

For example, if a user wants to run in batch-file mode with a file named *scenario1*, wants to enable collision detection for the simulation, wants to input a random seed of 1000, and wants to generate message tracing during program execution, the following will be entered on the command line:

-in scenario1 -c -rseed 1000 -m

or

-c -m -rseed 1000 -in scenario1

To stop the execution of the program, type in ⌘-d.

Running the Simulation as a Stand-Alone Program on the Apple Macintosh

Like any Macintosh Application, the simulation can be started by double-clicking on the application's icon, or by selecting the icon, and then selecting OPEN from the FILE menu.

3.6 Sample Output

In this section, we provide some sample output representative of the simulations capabilities. The output was generated in interactive mode, with redirected stdin (i.e., the psync.sens.test file), and the -tab option; the resulting table produced by the simulator was read into a charting program (Cricket Graph), to produce the graphs shown in Figure 11 and Figure 12.

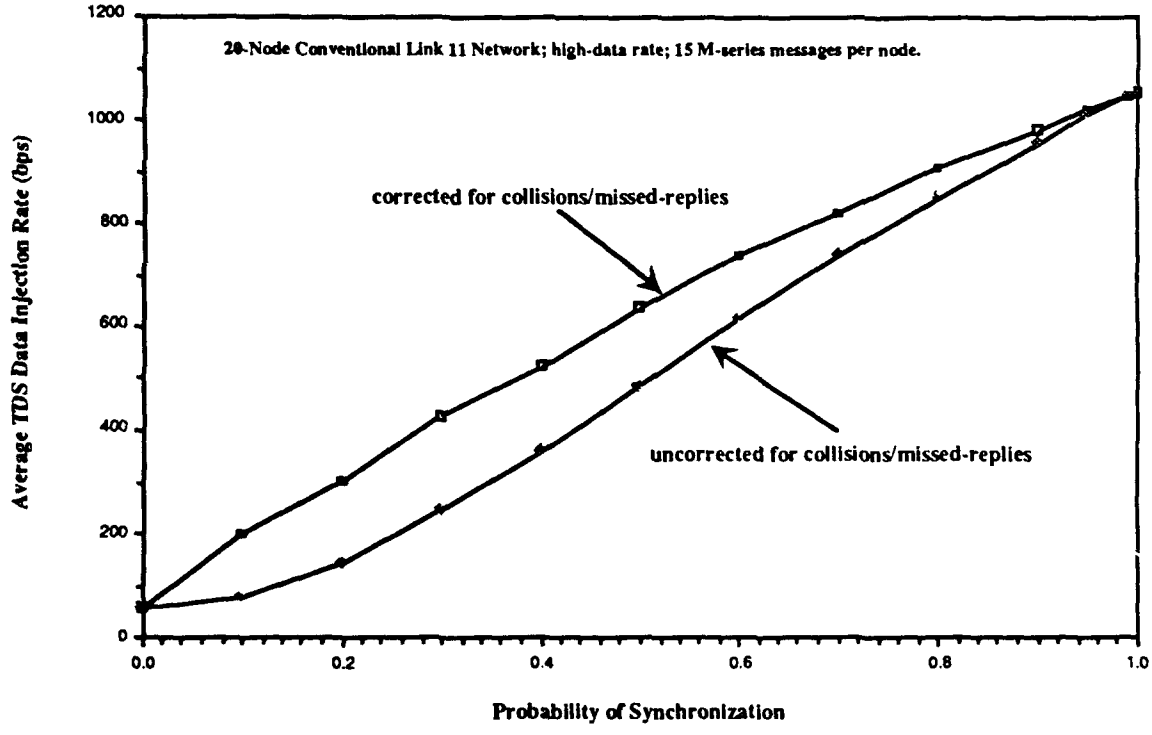

Figure 11: Average injected traffic rate, corrected for missed-replies and collision effects.(all waveform performance parameters assumed perfect, while synchronization performance varies).

As can be seen from Figure 11, the injected traffic rate increases when we account for missed-replies and collision effects; this is obviously true, since we are counting traffic that was sent (by the PU whose reply the DNCS missed) that we were not counting previously. This represents an improvement that would arise even if there were no other effects.

There is another real effect, however, on the net-cycle-frame length that occurs. The effect is illustrated in Figure 12, and is a decrease in the net cycle frame length when we account for collisions. The decrease in net-cycle frame length arises because the colliding interrogation cannot possibly generate a reply that would lengthen the cycle, assuming that PU replies are longer than a timeout period and interrogation, which they are, generally. In the probabilistic model, the interrogation could generate a reply that, again, assuming it was longer than a timeout period and interrogation, would lengthen the net cycle time. The collision effect violates the assumption of state-independence made in the probabilistic analysis; the simulation permits study of non-Markovian state transitions, where state transitions depend on the past history of the system (i.e., on collisions in previous states). Consequently, the decrease in net-cycle frame length which occurs with collision effects also contributes to the apparent increase in injected throughput, since more bits (the replies we weren't counting previously) are sent in a shorter net cycle.

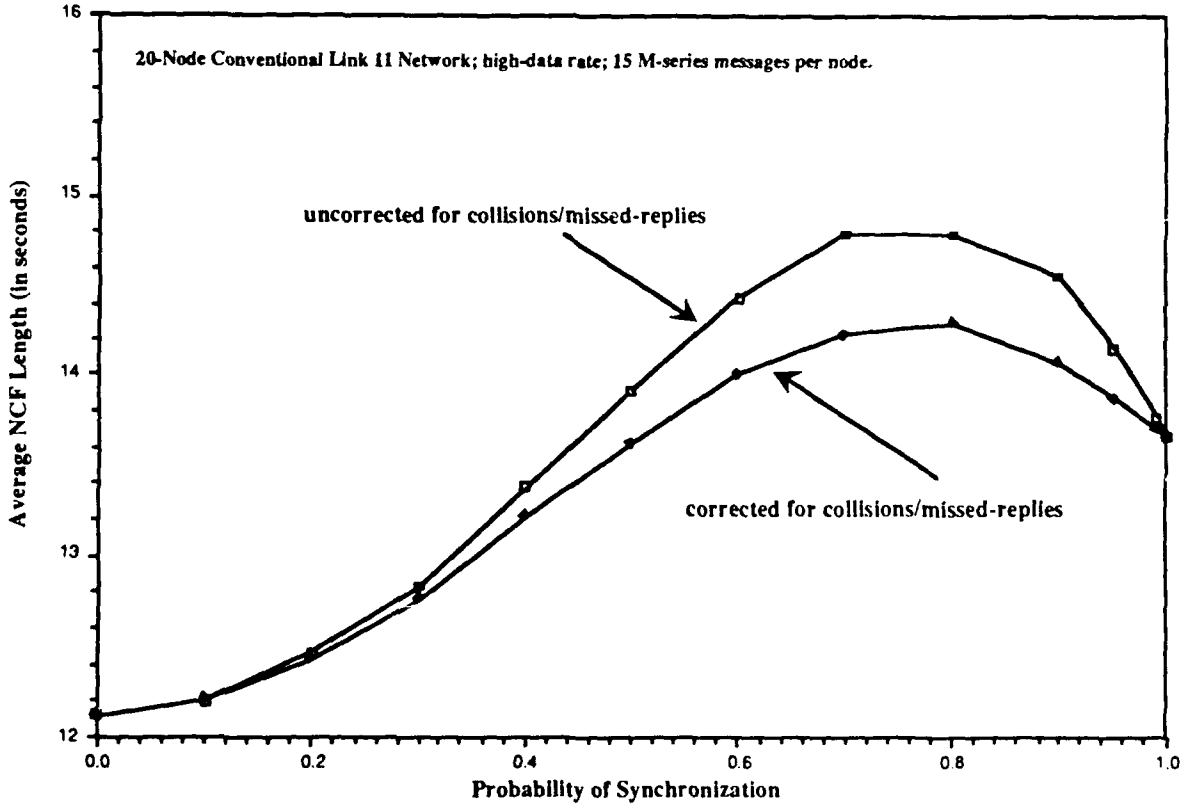

Figure 12. Average net-cycle-frame length, corrected for missed-replies and collision effects. (all waveform performance parameters assumed perfect, while synchronization performance varies).

4. SUMMARY

The simulation described here is intended as a tool for examining performance tradeoffs in waveform and network design parameters for the Link-11 Tactical Data Link used by the US. Navy and NATO. The simulation has been developed on an Apple Macintosh computer, using C and standard operating system libraries typical of UNIX workstation; the software is structured with the intent of porting the simulation to a UNIX workstation, though this has not yet been done. The simulation is an event driven simulation that can be used to study time-varying scenarios. We believe that it is a simple and useful tool for studying a wide range of tradeoffs related to Link-11 platform deployment and waveform design issues.

5. REFERENCES

(a) MIL-STD-188-203-1A, 31 DECEMBER 1986 (DRAFT)

(b) MILES System Implementation TYPE A Specification, Rockwell/Marconi Joint Venture Team, 26 July 89

(c) John S. Schlorff, "A Probabilistic Model for Link-11 Networking Operation", Communications Systems Branch, Information Technology Division, Naval Research Laboratory, NRL Memorandum Report 6776, Jan 25, 1991.

APPENDIX A: BATCH-MODE INPUT FILE FORMAT

INTRODUCTION

The input file contains all data required to run the simulation program. It contains the number of participating units (PUs) in the network, the number of net cycles in the simulation, each node's traffic load, and the waveform performance data for the links between each node. Those statistical data inciude probabilities of detecting the synchronization preamble, the probabilities of detecting the start-of-message codes, the probabilities of detecting the end-of-message codes, the probabilities of detecting a PU address, the probabilities of detecting the crypto-synchronization preamble (i.e, the message indicator), the probabilities of correctly receive Link 11 messages from another node, and the propagation delays between the receiving and the sending nodes. All of the statistical data are grouped into a link-parameter matrix, and the traffic-load data are arranged a the traffic-load matrix.

In addition, to model time-varying scenarios, the input file contains the number of times that nodal data are changed during the simulation. There are 3 types of changes allowed: type 1 in which only data of in link-parameter matrix is changed, type 2 in which only data in the traffic-load matrix is changed, and type 3 in which data in both the link-parameter matrix and the traffic-load matrix are changed.

FORMAT

The input file is created in the following format:

Line 1: number of participating units (PUs) in the network.
Line 2: number of net-cycle-frames (NCFs) that the program should run during the simulation
Line 3: number of times that the input data is changed.
Line 4: the link-parameter data matrix. In this matrix, there shall be NxN rows where N is the number of PUs in the network.Each row of the matrix shall be for data of each pair of nodes (a, b). The first line of matrix shall be for the pair of nodes (0,0), the second line shall be for the pair of nodes (0,1), the third line shall be for nodes (0,2), and so on in that order to cover all possible pairing combination of all PUs in the network. In this matrix, there shall be 8 columns in the following order from left to right: pair of node-id, probability of detecting the synchronization, probability of detecting the start of message, probability of detecting the end of message, probability of detecting its address, probability of detecting the message indicator, probability of correctly receive message, and propagation delay between two nodes. NOTE: Do not use TAB to separate data columns, use space-line only. So, data on the second line of the matrix and under the third column shall be the probability that node 1 can detect the start of message sending from node 0 because the second line is for the pair of nodes (0,1), and the third column is for the probability of detecting the start of message. If the number of rows in this matrix is less than NxN rows, the "Insufficient Data" error will appear.
Line 5: the traffic-load data matrix. In this matrix, there shall be two columns: one to tell the node IDs, and the other to tell the number of tracks held at that node. There shall be N rows in this matrix where each row shall correspond to one PU. The first row shall be for node 0, the second row shall be for node 1, and so on. Again, do not use TAB to separate the columns of data.

If there are changes in data input during the simulation, the following shall be added for each change:

Line 6: specific time that the change should occur. This time is measured from the beginning of the simulation.

Line 7: type of change (1, 2, or 3)

Line 8: new data matrix. If the type of change is 1, then the new data matrix shall be the new link-parameter data matrix. If the type of change is 2, then the new data matrix shall be the new traffic-load data matrix. If the type of change is 3, then the new data matrix shall be the new link-parameter data matrix and the new traffic-load data matrix.

NOTE: Comments can be inserted throughout the input file. However, they must be enclosed between two pound (#) signs. There is no space between the # sign and the adjacent character of the comment; e.g., #This is a comment#.

SAMPLE FILE

A sample of an input file is shown below for a network of 5 nodes. The data changes occur three times during the simulation:

```
# Number of pu:#
5                                                    #Line 1#

# Number of ncf:#
100                                                  #Line 2#

# Number of change:#
3                                                    #Line 3#

#The link parameter matrix which includes prob_sync,prob_som_detect,
prob_eom_detect, pro_add_detect, prob_mi_detect, prob_correct_mess, propa_delay.
The first row is for node (0,0), the second row is for node (0,1),
and so on.

node   p_syn       p_som      p_eom      p_add      p_mi       p_cr_me    p_del#

 (0,0)  1.000000  1.000000  1.000000  1.000000  1.000000  1.000000  1.000000     #Line 4#
(0,1)  0.780000  0.780000  0.780000  0.780000  0.780000  0.780000  0.780000
(0,2)  0.975000  0.975000  0.975000  0.975000  0.975000  0.975000  0.975000
 (0,3)  0.650000  0.650000  0.650000  0.650000  0.650000  0.650000  0.650000
(0,4)  0.078000  0.078000  0.078000  0.078000  0.078000  0.078000  0.078000
(1,0)  1.0   1.0        1.0        1.0        1.0        1.0        1.0
(1,1)  1.0   1.0        1.0        1.0        1.0        1.0        1.0
              ....
              ....

#Traffic loading matrix which includes number of track held per node.
The first row is for node 0, second row is for node 1, and so on.

node                number of tracks#

0                   5                                #Line 5#
```

1 *8*
2 *9*
3 *16*
4 *25*

#change time :#
15 *#Line 6#*

#change type :#
3 *#Line 7#*

#Since the type of change is 3, the new data matrix will be the new link parameter matrix and the new traffic load matrix#

#The link parameter matrix which includes prob_sync,prob_som_detect, prob_eom_detect, pro_add_detect, prob_mi_detect, prob_correct_mess, propa_delay. The first row is for node (0,0), the second row is for node (0,1), and so on.

node	*p_syn*	*p_som*	*p_eom*	*p_add*	*p_mi*	*p_cr_me*	*p_del#*	
(0,0)	*1.0*	*1.0*	*1.0*	*1.0*	*1.0*	*1.0*	*1.0*	*#Line 8#*
(0,1)	*1.0*	*1.0*	*1.0*	*1.0*	*1.0*	*1.0*	*1.0*	
(0,2)	*1.0*	*1.0*	*1.0*	*1.0*	*1.0*	*1.0*	*1.0*	
(0,3)	*1.0*	*1.0*	*1.0*	*1.0*	*1.0*	*1.0*	*1.0*	
(0,4)	*1.0*	*1.0*	*1.0*	*1.0*	*1.0*	*1.0*	*1.0*	
(1,0)	*0.975000*	*0.975000*	*0.975000*	*0.975000*	*0.975000*	*0.975000*	*0.975000*	
(1,1)	*0.650000*	*0.650000*	*0.650000*	*0.650000*	*0.650000*	*0.650000*	*0.650000*	

....
....

#The traffic loading matrix which includes number of track held per node. The first row is for node 0, second row is for node 1, and so on.

node	*number of tracks#*
0	*6*
1	*8*
2	*9*
3	*14*
4	*15*

#Line 6, Line 7, and Line 8 shall be added for each change#

#change time:#
20 *#Line 6#*

#change type:#
2 *#Line 7#*

#Since the type of change is 2, the new data matrix is the new traffic-load data matrix#

#The traffic loading matrix which includes number of track held per node. The first row is for node 0, second row is for node 1, and so on.

node	*number of tracks#*	
0	*5*	*#Line 8#*
1	*8*	
2	*9*	
3	*16*	
4	*25*	

#change time:#
22.87

#change type:#
1

#Sine the type of change is 1, the new data matrix is the new link parameter data matrix#

#The link parameter matrix which includes prob_sync,prob_som_detect, prob_eom_detect, pro_add_detect, prob_mi_detect, prob_correct_mess, propa_delay. The first row is for node (0,0), the second row is for node (0,1), and so on.

	node	*p_syn*	*p_som*	*p_eom*	*p_add*	*p_mi*	*p_cr_me*	*p_del#*
(0,0)	*1.000000*	*1.000000*	*1.000000*	*1.000000*	*1.00000*	*1.000000*	*1.000000*	
(0,1)	*0.780000*	*0.780000*	*0.780000*	*0.780000*	*0.78000*	*0.780000*	*0.780000*	
(0,2)	*0.975000*	*0.975000*	*0.975000*	*0.975000*	*0.97500*	*0.975000*	*0.975000*	
(0,3)	*0.650000*	*0.650000*	*0.650000*	*0.650000*	*0.65000*	*0.650000*	*0.650000*	
(0,4)	*0.078000*	*0.078000*	*0.078000*	*0.078000*	*0.078000*	*0.078000*	*0.078000*	
(1,0)	*1.0*	*1.0*	*1.0*	*1.0*	*1.0*	*1.0*	*. 1.0*	
(1,1)	*1.0*	*1.0*	*1.0*	*1.0*	*1.0*	*1.0*	*. 1.0*	
....								
....								